S224309

System Programming, SYSPROG3

Javad Malek Shahkoohi

Q1:

```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <errno.h>
#include <sys/types.h>
#include <unistd.h>
#include <pthread.h>
#include <sys/time.h>

/* if you don't want to manage the errors and exception you can reboot your PC, (also you cannot print the result of your computation)
I prefer to control this logical error and produce the result of my program at time of error
*/
void *PrintHello(void *threadid)
{
   pthread_exit(NULL);
}

void main()
{
	int pid1, pid2, rc;
	int countrProcesses=2, countrThreads=2;

	struct timeval start, end;
	long mtime, seconds, useconds;
	gettimeofday(&start, NULL);

	printf("\n *************** Starting to Fork And Execute ************ \n\n");
	pid1 = fork();
	switch(pid1)
	{
		case -1:
			printf("Error in fork Function\n\n\n");
			break;
		case 0: // First Child of Fork 2 counts processes
			//int countrProcesses=2; // parent and current child
			while(1) // Infint loop to fork new processes by First Child
			{
				pid2 = fork();
				switch(pid2)
				{
					case -1:  // printing the result when it is impossible to create new processe (because it is over)
						gettimeofday(&end, NULL);
```

S224309

System Programming, SYSPROG3

Javad Malek Shahkoohi

```
                                                seconds  = end.tv_sec  - start.tv_sec;
                                                useconds = end.tv_usec - start.tv_usec;
                                                mtime = ((seconds) * 1000 + useconds/1000.0) +
0.5; // computing the time difference at begging of program up the time of last process
                                                printf("Processes Elapsed time: %ld
milliseconds\n\n", mtime);
                                                printf("MAX CHILD ID IS
:%ld\n\n",sysconf(_SC_CHILD_MAX));

                                                printf("countrProcesses =
%d\n\n\n\n\n\n",countrProcesses);
                                                return;
                                        case 0: // Doing somthing in second child
                                                exit(0);
                                                break;
                                        default: // counting the number of processes
                                                countrProcesses++;
                                                break;
                                };
                        } // End Of Processes While
                        break;

                default: // First Parent 2 counts Threads
                        //int countrThreads=2; // parent and current child
                        while(1)
                        {
                                pthread_t tmpThrd; int i;
                                rc = pthread_create(&tmpThrd, NULL, PrintHello, (void *)i);
                                if (rc)    {
                                        gettimeofday(&end, NULL);
                                        seconds  = end.tv_sec  - start.tv_sec;
                                        useconds = end.tv_usec - start.tv_usec;
                                        mtime = ((seconds) * 1000 + useconds/1000.0) + 0.5;
                                        printf("Threads Elapsed time: %ld milliseconds\n",
mtime);

                                        printf("countrThreads = %d\n\n\n\n\n\n",countrThreads);
                                        pthread_exit(NULL);
                                        return;
                                }
                                else countrThreads++;

                        }// End Of Threads While
                        break;
        }
}
```

Q2:

S224309

System Programming, SYSPROG3

Javad Malek Shahkoohi

```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <errno.h>
#include <sys/types.h>
#include <unistd.h>
#include <pthread.h>
#include <sys/time.h>


/*
int Counter = 0;
it had the concurrency problem when the CounterThread thread tried to change Counter and the solution is to use the mutex
*/

long Counter = 0;
void *Conter() //infinit increment function for increments thread
{
	while(1)
	{
		Counter++;
		//printf("Counter1 = %ld\t", Counter);
	}
}

void *Printer() // printing the value of counter in the infinit loop by a second sleep
{
	while(1)
	{
		printf("Counter2 = %ld\n", Counter);
		sleep(1);
	}
}


void main()
{
	int pid1, pid2, rc1, rc2;
	pthread_t ConterThrd,PrinterThrd;

	rc1 = pthread_create(&ConterThrd, NULL, Conter, NULL);
	if (rc1)  {
		printf("Error:unable to create thread, error no: %d", rc1);
		return;
	}
```

S224309
System Programming, SYSPROG3
Javad Malek Shahkoohi

```
	rc2 = pthread_create(&PrinterThrd, NULL, Printer, NULL);
	if (rc2)  {
		printf("Error:unable to create thread, error no: %d", rc2);
		return;
	}

	pthread_join( ConterThrd, NULL);
	pthread_join( PrinterThrd, NULL);
}
```

Q3:

```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <errno.h>
#include <sys/types.h>
#include <unistd.h>
#include <pthread.h>
#include <sys/time.h>
#include <string.h>
#include <sys/stat.h>

/*
this is the Writer&Reader problem because there is a common buffer to read and write,
here I assumed the limit for the length of file
and reader starts to read first then the writer will start its jobs
*/
int length=0;
char buffer[10000];

void *Reader(void *argv_ReadFileStr)
{
	int file;
	file = open(argv_ReadFileStr, O_RDONLY);
	length = read(file,buffer, 10000);
	printf("\nlength: %d\n",length);

}

void *Writer(void *argv_WriteFileStr)
{
	int file;
	file = open(argv_WriteFileStr, O_WRONLY|O_CREAT, 0644);
	write(file,buffer, length);
}
```

S224309
System Programming, SYSPROG3
Javad Malek Shahkoohi

```
void main(int argc, char *argv[])
{
	if(argc != 3) {
		printf("Error in input data (Read file, write file name)");
		return;
	}

	pthread_t ReaderThrd,WriterThrd;

	pthread_create(&ReaderThrd, NULL, Reader, (void *)argv[1]); // calling the reader thread to read from file
	pthread_join( ReaderThrd, NULL); // the main thread wait for child thread to complete the its job

	pthread_create(&WriterThrd, NULL, Writer, (void *)argv[2]); //calling the writer thread to write into second file
	pthread_join( WriterThrd, NULL);
}
```

Q4:

```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <errno.h>
#include <sys/types.h>
#include <unistd.h>
#include <pthread.h>
#include <sys/time.h>

/*
the only a semaphore is not enough to protect the mutex because the turn is important , hence the conditional variable is necessary to protect that
*/
pthread_mutex_t condition_mutex = PTHREAD_MUTEX_INITIALIZER;  // a semaphore for mutex
pthread_cond_t  condition_cond  = PTHREAD_COND_INITIALIZER;
/* a conditional variable (because at the first time when a thread comes and it is not its turn, this thread must release the mutex and wait for other thread to do its job)*/
int turn=0;
long Counter = 0;

void *Conter()
{
	while(1)
	{
		pthread_mutex_lock( &condition_mutex ); // choon momken hast meghdari baraye khoondan nabashad
		if(turn==0) pthread_cond_wait( &condition_cond, &condition_mutex );
```

S224309
System Programming, SYSPROG3
Javad Malek Shahkoohi

```
                turn=1;
                Counter++;  // changes the shared variable

                pthread_cond_signal( &condition_cond );
                pthread_mutex_unlock( &condition_mutex );
        }
}

void *Printer()
{
        while(1)
        {
                pthread_mutex_lock( &condition_mutex );
                if(turn==1) pthread_cond_wait( &condition_cond, &condition_mutex );

                turn=1;
                printf("Counter2 = %ld\n", Counter);
                sleep(1);

                pthread_cond_signal( &condition_cond );
                pthread_mutex_unlock( &condition_mutex );
        }
}


void main()
{
        int pid1, pid2, rc1, rc2;
        pthread_t ConterThrd,PrinterThrd;

        rc1 = pthread_create(&ConterThrd, NULL, Conter, NULL);
        if (rc1)  {
                printf("Error:unable to create thread, error no: %d", rc1);
                return;
        }
        rc2 = pthread_create(&PrinterThrd, NULL, Printer, NULL);
        if (rc2)  {
                printf("Error:unable to create thread, error no: %d", rc2);
                return;
        }

        pthread_join( ConterThrd, NULL);
        pthread_join( PrinterThrd, NULL);
}
```

Q5:

S224309

System Programming, SYSPROG3

Javad Malek Shahkoohi

```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <errno.h>
#include <sys/types.h>
#include <unistd.h>
#include <pthread.h>
#include <sys/time.h>

/*
int shareValue = 10;
it had the concurrency problem when each thread tried to change shareValue and the solution is
to use the mutex
*/
int shareValue = 10;
void *F1() // for the second child to call for its threads
{
        printf("\n\nin PID=%d 1st shareValue = %d",getpid(),shareValue);
        //shareValue = 1;
        printf("\t new shareValue = %d\n",shareValue);
        pthread_exit(NULL);
}
void *F2()// for the first child to call for its threads
{
        printf("\n\nin PID=%d 2st shareValue = %d",getpid(),shareValue);
        //shareValue = 2;
        printf("\t PID=%d new shareValue = %d\n",shareValue);
        pthread_exit(NULL);
}
void *F3()// for the main process to call for its threads
{
        printf("\n\nin PID=%d 3st shareValue = %d",getpid(),shareValue);
        //shareValue = 3;
        printf("\t PID=%d new shareValue = %d\n",shareValue);
        pthread_exit(NULL);
}

void main()
{
        int pid1, pid2;
        pthread_t T1,T2,T3;

        shareValue=0;

        pid1 = fork();
        switch(pid1)
        {
```

S224309

System Programming, SYSPROG3

Javad Malek Shahkoohi

```
		case -1:
			printf("Error in forck Function\n\n\n");
			break;
		case 0:
			//execl("/bin/ls","bin/ls -l",(char *)0,0);
			/* runs the command but the other processes terminated!? */
			pid2 = fork();
			//execl("/bin/ls","bin/ls -l",(char *)0,0);
			switch(pid2)
			{
				case -1:
					printf("Error in forck Function\n\n\n");
					exit(0);
				case 0: //second child
					//execl("/bin/ls","bin/ls -l",(char *)0,0);
					/* runs the command but sometimes the other processes
terminated! and sometimes they runned */
					pthread_create(&T1, NULL, F1, NULL);
					pthread_create(&T2, NULL, F1, NULL);
					pthread_create(&T3, NULL, F1, NULL);
					//execl("/bin/ls","bin/ls -l",(char *)0,0);
					pthread_join( T1, NULL);
					pthread_join( T2, NULL);
					pthread_join( T3, NULL);
					exit(0);
					break;
				default: //first child
					pthread_create(&T1, NULL, F2, NULL);
					pthread_create(&T2, NULL, F2, NULL);
					pthread_create(&T3, NULL, F2, NULL);
					pthread_join( T1, NULL);
					pthread_join( T2, NULL);
					pthread_join( T3, NULL);
					break;
			}
			break;

		default: //father
			pthread_create(&T1, NULL, F3, NULL);
			pthread_create(&T2, NULL, F3, NULL);
			pthread_create(&T3, NULL, F3, NULL);
			pthread_join( T1, NULL);
			pthread_join( T2, NULL);
			pthread_join( T3, NULL);

			break;
	}
```

S224309

System Programming, SYSPROG3

Javad Malek Shahkoohi

}

Q6:

```
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>


int GlobalCounter = 0, NewNumber = 0;;
pthread_mutex_t mutexsum;

void *Func1(void *threadid)   // for first thread to take value and starting the programm, other threads MUST WAITE for initializing
{
	int taskId;
	taskId = (int)threadid;

	pthread_mutex_lock (&mutexsum);
	printf("\n\nThread 1: Enter New Number To be added the Counter (0: Terminate the process)\n");
	scanf("%d",&NewNumber);
	GlobalCounter += NewNumber;
	printf("taskId: %d ,\tThread 1: GlobalCounter is %d\n", taskId,GlobalCounter);
	pthread_mutex_unlock (&mutexsum);

	pthread_exit(NULL);
}

void *Func234(void *threadid)
{
	int taskId;
	taskId = (int)threadid;
	int TID = pthread_self();

	while (GlobalCounter == 0);// printf("\nwait please...");;
/*	this is a busy waiting and the solution is conditional variable (there is turn problem and initalization problem at first-
 it is possible for other 3rd threads to read when the value has no correct value)*/

	while(GlobalCounter >= 0)
	{
		GlobalCounter--;
		printf("taskId: %d ,\tThread ID: %d ,\t GlobalCounter is %d\n", taskId,TID, GlobalCounter);
		sleep(1);
		pthread_mutex_unlock (&mutexsum);
	}
```

```
	pthread_exit(NULL);
}

void main(int argc, char *argv[])
{
	pthread_t threads[4];
	pthread_attr_t attr;
	void *status;
	int rc, t;

	pthread_mutex_init(&mutexsum, NULL);
	pthread_attr_init(&attr);
	pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_JOINABLE);

	t = 0;	printf("Creating thread : %d\n",t);
	pthread_create(&threads[t], &attr, Func1, (void *)t);

	t = 1;	printf("Creating thread : %d\n",t);
	pthread_create(&threads[t], &attr, Func234, (void *)t);

	t = 2;	printf("Creating thread : %d\n",t);
	pthread_create(&threads[t], &attr, Func234, (void *)t);

	t = 3;	printf("Creating thread : %d\n",t);
	pthread_create(&threads[t], &attr, Func234, (void *)t);

	pthread_attr_destroy(&attr);
	for(t=0;t<4;t++)
		pthread_join(threads[t], &status);

	pthread_mutex_destroy(&mutexsum);
	pthread_exit(NULL);
}
```

Q7:

```
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>

#define NUM_THREADS10
#define LEN_ARRAY_THREADS   10
int GlobalGeneratedValues[LEN_ARRAY_THREADS*NUM_THREADS];
pthread_mutex_t mutexsum;

void *GenerateValues(void *threadid)
{
```

S224309

System Programming, SYSPROG3

Javad Malek Shahkoohi

```
	int taskId;
	taskId = (int)threadid;
	//printf("\taskId ,  %d\n", taskId);

	int i=0, tmepvalue;
	int LocalThreadGeneratedValues[NUM_THREADS];

	for (i =0; i <= LEN_ARRAY_THREADS; i++)		// generates the random values and put in the local array in side the thread
	{
		tmepvalue = rand() % 11; // between 0 and 10
		while (tmepvalue == 0 )
			tmepvalue = rand() % 11;
		LocalThreadGeneratedValues[i] = tmepvalue;
	}

	int SegmentNo = (LEN_ARRAY_THREADS*taskId); // buliding the index offset of each thread to work with array
	pthread_mutex_lock (&mutexsum);		// starting the critical section
	for(i=0; i < LEN_ARRAY_THREADS; i++)
	{
		GlobalGeneratedValues[SegmentNo + i] = LocalThreadGeneratedValues[i];
		printf("\ntaskId:%d , GlobalGeneratedValues[%d] = %d \t",taskId,i, LocalThreadGeneratedValues[i]);
	}
	pthread_mutex_unlock (&mutexsum);

	pthread_exit(NULL);
}

void main(int argc, char *argv[])
{
	pthread_t threads[NUM_THREADS];
	pthread_attr_t attr;
	void *status;
	int rc, t;

	pthread_mutex_init(&mutexsum, NULL);
	pthread_attr_init(&attr);
	pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_JOINABLE);

	for(t=0;t<NUM_THREADS;t++)
	{
		printf("Creating thread : %d\n", t);

		rc = pthread_create(&threads[t], &attr, GenerateValues, (void *)t);
		if (rc) {  printf("ERROR; return code from pthread_create() is %d\n", rc);  exit(-1);}
```

```
            }

        pthread_attr_destroy(&attr);
        for(t=0;t<NUM_THREADS;t++)
        {
                rc = pthread_join(threads[t], &status);
                if (rc){    printf("ERROR return code from pthread_join() is %d\n", rc);       exit(-1);
        }
        }

        int sum = 0;       // make the summation at the end, when all the threads have been
finished
        for(int i=0; i<LEN_ARRAY_THREADS*NUM_THREADS; i++)
        {
                printf("\nGlobalGeneratedValues[%d] = %d \t Sum = %d",i,
GlobalGeneratedValues[i], sum);
                sum += GlobalGeneratedValues[i];
        }
        printf("\n\n\n\n The sum of all generated value numbers by threads is : %d",sum);

        pthread_mutex_destroy(&mutexsum);
        pthread_exit(NULL);
}
```

Q8:

```
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>


long GlobalSumValues = 0;
int GlobalX = 0;
pthread_mutex_t mutexsum;

void *Xpower2I(void *threadid)
{
        int taskId;
        taskId = (int)threadid;


        long i=0, tmepvalue = 1;
        int LocalPower=0;

        for (i =0; i < taskId; i++)  // compute the 2^i by each thread
        {
                tmepvalue *= GlobalX;
        }
```

S224309
System Programming, SYSPROG3
Javad Malek Shahkoohi

```
	pthread_mutex_lock(&mutexsum);  // tyrs to add to the globl sum
	GlobalSumValues += tmepvalue;
	printf("\ntaskId:%d ,\tLocalSumValue = %ld,\tGlobalSumValues = %ld ",taskId,tmepvalue,
GlobalSumValues);
	pthread_mutex_unlock(&mutexsum);


	pthread_exit(NULL);
}

void main(int argc, char *argv[])
{
	if(argc != 3){	printf("\nInput arrguments are not correct....\n");	return; }
	GlobalX = atoi(argv[1]);
	int NUM_THREADS = atoi(argv[2]);

	pthread_t threads[NUM_THREADS];
	pthread_attr_t attr;
	void *status;
	int rc, t;

	pthread_mutex_init(&mutexsum, NULL);
	pthread_attr_init(&attr);
	pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_JOINABLE);

	for(t=0;t<NUM_THREADS;t++)
	{
		printf("Creating thread : %d\n", t);

		rc = pthread_create(&threads[t], &attr, Xpower2I, (void *)t);
		if (rc) {  printf("ERROR; return code from pthread_create() is %d\n", rc);  exit(-1);}
	 }

	pthread_attr_destroy(&attr);
	for(t=0;t<NUM_THREADS;t++)
	{
		rc = pthread_join(threads[t], &status);
		if (rc){   printf("ERROR return code from pthread_join() is %d\n", rc);	exit(-1);
	}
	}

	printf("\n\n\nGlobalSumValues = %ld \n",GlobalSumValues);

	pthread_mutex_destroy(&mutexsum);
	pthread_exit(NULL);
}
```

S224309

System Programming, SYSPROG3

Javad Malek Shahkoohi

Q9:

```
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>

int IsPrime = 1;
int Golba_N = 0; // the value to check prime
int Global_P = 0; // the number of threads
int SubSetLen = 0;   // sub length to check by thread
int RemainedThread = 0;
/*
here we divided SubSetLen = Golba_N/Global_P; to find the length of compution by each thread
*/

pthread_mutex_t mutexsum;

void *CheckPrime(void *threadid)
{
	int taskId, i=0;
	taskId = (int)threadid;

	int startIndex = (SubSetLen * taskId);  // the offset to startIndex
	if(taskId == 0) i=2; // the first task must skip 1,2
	/*
	(Golba_N/2) >= (startIndex + i) to avoid over computation when we reach to the N/2
	(startIndex + i) != 1; when the length is 1 all the time Golba_N % 1 = 0 then the
computation ahead on wrong way
	*/
	for(; i<SubSetLen && (Golba_N/2) >= (startIndex + i) && (startIndex + i) != 1; i++ )
	{
		if(Golba_N % (startIndex + i) == 0)
		{
			pthread_mutex_lock(&mutexsum);
			IsPrime = 0;
			printf("taskId:%d ,\tIsnot Prime: %d/%d=0\n",taskId,Golba_N,(startIndex
+ i));
			pthread_mutex_unlock(&mutexsum);return;
			pthread_exit(NULL);
		}
	}

	/*// other semaphore to count the number of processes because when the number is not
prime, we can find the end of computation*/
	pthread_mutex_lock(&mutexsum);
	RemainedThread--;
	printf("taskId:%d ,\tRemainedThread = %d \n",taskId,RemainedThread);
```

```
	pthread_mutex_unlock(&mutexsum);

	pthread_exit(NULL);
}

void main(int argc, char *argv[])
{
	if(argc != 3){	printf("\nInput arrguments are not correct Global_P Golba_N ...\n");
	return; }
	Global_P = atoi(argv[1]);
	RemainedThread = Global_P;
	Golba_N = atoi(argv[2]);

	SubSetLen = Golba_N/Global_P; // tool subset har thread
	if(SubSetLen == 0)
	{
		printf("\nThe number of threads are more than N, Global_P must be less than
Golba_N because it is impossible to divid the elements among threads\n");
		return;
	}

	pthread_t threads[Global_P];
	pthread_attr_t attr;
	int rc, t;

	pthread_mutex_init(&mutexsum, NULL);

	for(t=0; t < Global_P; t++)
	{
		printf("Creating thread : %d\n", t);

		rc = pthread_create(&threads[t], NULL, CheckPrime, (void *)t);
		if (rc) {  printf("ERROR; return code from pthread_create() is %d\n", rc);  exit(-1);}
	}

	/*the unlimited loop up to the time one of threads changes the value of IsPrime = 0 Or
one of threads remained
	Isprime checked by main thread continuosly */
	while(1 ) //  ta zamanike yeki az thread ha IsPrime ra 0 kond ya hanooz threadei baghi
mandeh bashad
	{
		if (IsPrime == 0) // Is not prime
		{
			printf("\n\nMain Function: Golba_N = %d isnot prime \n", Golba_N);
			pthread_mutex_destroy(&mutexsum);
			pthread_exit(NULL);
			return;
```

S224309

System Programming, SYSPROG3

Javad Malek Shahkoohi

```
                }

                if(RemainedThread == 0)          // no one of threads changed the value of IsPrime
and all of them terminated completely
                {
                        printf("\n\nMain Function: Golba_N = %d is prime \n", Golba_N);
                        pthread_mutex_destroy(&mutexsum);
                        pthread_exit(NULL);
                        return;
                }
        }
}
```